**Alexa Progressive response using python**

**See** [**https://github.com/jallwork/Alexa\_Progressive\_Response**](https://github.com/jallwork/Alexa_Progressive_Response)

A progressive response is used to tell your user that your program is working on something (e.g. a taxi ride, or pizza delivery).

**Steps to Send a Progressive Response** (Amazon docs)

<https://developer.amazon.com/en-US/docs/alexa/custom-skills/send-the-user-a-progressive-response.html#steps-to-send-a-progressive-response>

To send a progressive response, call the Progressive Response API and send a directive:

* Get the required data from the incoming request (LaunchRequest or IntentRequest). You need the apiAccessToken and requestId to construct a valid Progressive Response API request.
* Call the Progressive Response API and send a directive (such as VoicePlayer.Speak) with the content. The content must be valid SSML wrapped in <speak> tags.
* Complete your normal skill processing.
* Once the progressive response call completes, return your full skill response object. Note that you cannot send any more progressive responses after you return the response object.

NodeJS example Dabble labs video <https://www.youtube.com/watch?v=-Vk9Ei6bT4g>

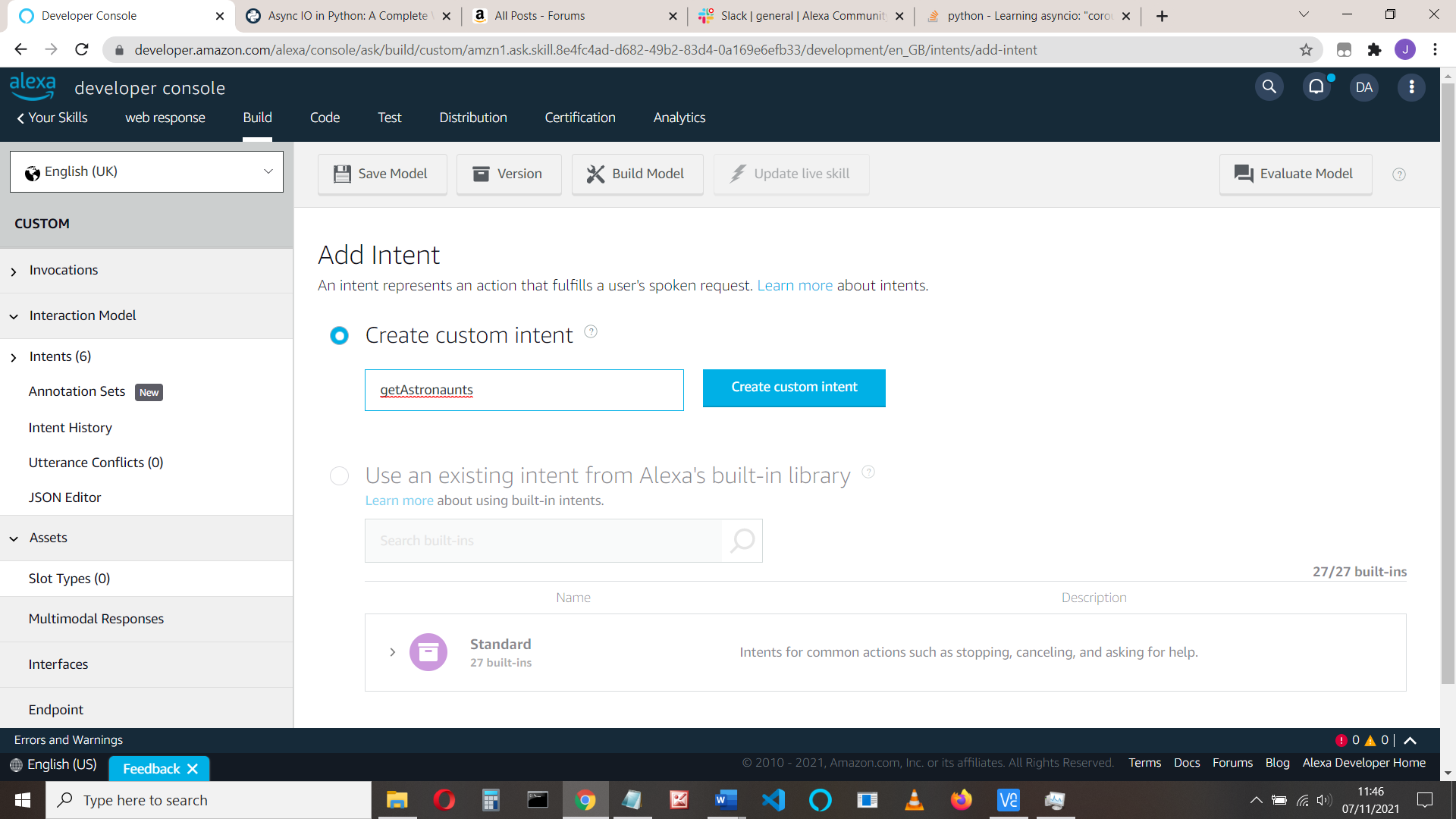
We’ll do similar in python.

You can just call the progressive response, then carry on with your code. We’ll do that, then access a web site, but add a delay to show how it works.

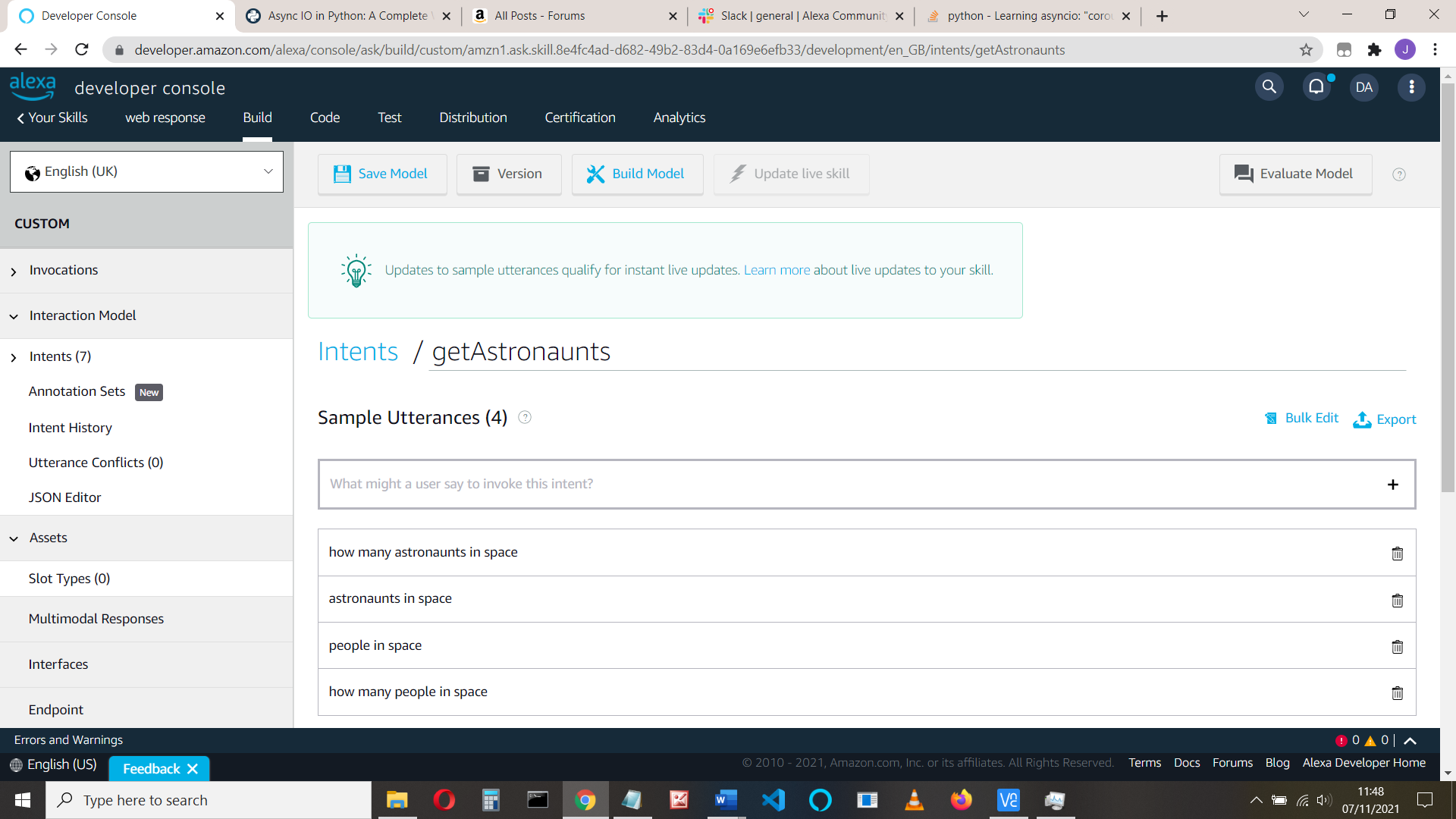
**Task:**

Start a new Python Alexa Hosted skill, start from scratch. Give it a suitable name (I’ve call it progressive response)

We’ll add a new intent that finds the number of people currently in space. Add a new intent, call ‘getAstronauts’



Add some utterances:



The json is:

{

"interactionModel": {

"languageModel": {

"invocationName": "progressive response",

"intents": [

{

"name": "AMAZON.CancelIntent",

"samples": []

},

{

"name": "AMAZON.HelpIntent",

"samples": []

},

{

"name": "AMAZON.StopIntent",

"samples": []

},

{

"name": "AMAZON.NavigateHomeIntent",

"samples": []

},

{

"name": "HelloWorldIntent",

"slots": [],

"samples": [

"hello",

"how are you",

"say hi world",

"say hi",

"hi",

"say hello world",

"say hello"

]

},

{

"name": "AMAZON.FallbackIntent",

"samples": []

},

{

"name": "getAstronauts",

"slots": [],

"samples": [

"how many astronauts in space",

"astronauts in space",

"people in space",

"how many people in space"

]

}

],

"types": []

}

}

}

Save and build the model

Click the code tab and edit *requirements.txt.* Add: **aiohttp == 3.8.0**

With the rest of the ‘froms’ add:

**from ask\_sdk\_model.services.directive import (**

**SendDirectiveRequest, Header, SpeakDirective)**

**from ask\_sdk\_core.skill\_builder import CustomSkillBuilder**

**from ask\_sdk\_core.api\_client import DefaultApiClient**

**sb = CustomSkillBuilder(api\_client=DefaultApiClient())**

Comment out the line at the end:

**#sb = SkillBuilder()**

Add the imports

**import aiohttp**

**import asyncio**

**import json**

**import time**

Add the functions we need:

See <https://pypi.org/project/aiohttp/>

**async def get(url):**

**time.sleep(5) # only here to provide long response**

**async with aiohttp.ClientSession() as session:**

**async with session.get(url) as resp:**

**return await resp.text()**

**def get\_progressive\_response(handler\_input):**

**# type: (HandlerInput) -> None**

**request\_id\_holder = handler\_input.request\_envelope.request.request\_id**

**directive\_header = Header(request\_id=request\_id\_holder)**

**speech = SpeakDirective(speech="Ok - wait")**

**directive\_request = SendDirectiveRequest(**

**header=directive\_header, directive=speech)**

**directive\_service\_client = handler\_input.service\_client\_factory.get\_directive\_service()**

**directive\_service\_client.enqueue(directive\_request)**

**return**

Change LaunchRequest speak\_output code to:

**speak\_output = "Hi, you can ask for how many people in space"**

Do the same for Hello world intent if you want

Add the getAstronauts intent code (say before HelloWorld intent handler):

**class GetAstronautsIntentHandler (AbstractRequestHandler):**

**# Handler for Astronaut Count Intent**

**def can\_handle(self, handler\_input):**

**# type: (HandlerInput) -> bool**

**return ask\_utils.is\_intent\_name ("getAstronauts")(handler\_input)**

# make sure your intent\_name agrees with what you called the intent in the build

**def handle(self, handler\_input):**

**speech\_text = "OK - wait"**

**try:**

**url = "http://api.open-notify.org/astros.json"**

**get\_progressive\_response(handler\_input)**

**htmlData = asyncio.run(get(url))**

**# extract data**

**jsonResponse = json.loads(htmlData)**

**people = jsonResponse.get('number')**

**speech\_text = "There are currently " + str(people) + " people in space"**

**except Exception as ex:**

**speech\_text = " An exception occurred"**

**return (**

**handler\_input.response\_builder**

**.speak(speech\_text) #(await promise)**

**#.ask(speech\_text)**

**.response**

**)**

Finally comment out the sb = that’s provided and add the intent handler to the skill builder at the end of your code:

**#sb = SkillBuilder()**

sb.add\_request\_handler(LaunchRequestHandler())

**sb.add\_request\_handler(GetAstronautsIntentHandler())**

Save, deploy, test, enable development and try your skill.

BUT the developer console **doesn’t execute progressive responses properly**, so try it on a device.

Remember your response still only has 8 seconds to do it all.

**How does it work? - Awaiting a web response**

We need to wait until our web resource responds with its data. To do this we use the **await** command. The await command MUST be in an **async** coroutine. We call the async coroutine using asyncio.run()

(To actually run a coroutine, asyncio provides three main mechanisms, see <https://docs.python.org/3/library/asyncio-task.html#running-tasks-concurrently> )

Use **await** command is used to pause your code and wait for your function to finish.The await only works inside **async** function.

The basic *async/await* is this:

async def afunction():

# Pause here and come back to afunction () when function2() is ready

r = await function2()

return r

This needs to be called using asyncio.run:

asyncio.run(afunction())

If some data is returned (as it the case above) you call it like this:

someData = asyncio.run(afunction ())

Note: you can’t just write: someData = afunction ()

It’ll error: RuntimeWarning: coroutine .. was never awaited

To wait on a web response, we use this code from: https://pypi.org/project/aiohttp/

async def get(url):

async with aiohttp.ClientSession() as session:

async with session.get(url) as resp:

return await resp.text()

and call it with:

htmlData = asyncio.run(get(url))

But before we call that, we send a progressive response to say wait while we’re doing it

You can run both the get() and the progressive\_response function concurrently by using: asyncio.gather (get(), progressive\_reponse())

Simples!

Appendix - JSON response

The web site responds with some json that looks (short form) like this:

We extract the number of people from this

{

"number": 10,

"people": [{

"craft": "ISS",

"name": "Mark Vande Hei"

},

{

"craft": "ISS",

"name": "Pyotr Dubrov"

},

{

"craft": "ISS",

"name": "Megan McArthur"

},

{

"craft": "Shenzhou 13",

"name": "Ye Guangfu"

}

],

"message": "success"

}